lab1 report

于新雨 计25 2022010841

基础功能

为了让我们 syscall_trace 里面可以方便的访问到当前任务调用编号为 id 的系统调用的次数，我们先是全局的实现了一个 trace_array 数组，每一项用来表示对应 syscall 的执行次数，并且实现了读，增加，清零的功能，在 `exit_current_and_run_next()` 我们需要将该 trace_array 清零

然后正常实现了 syscall_trace 的读入写入功能

简答作业

1. 在 ch2b_bad_instructions.rs 中，用了 S mode 特权指令 `sret`。在 ch2b_bad_register.rs 中，用了 S mode 指令 `csrr` 尝试读取 sstatus 的值写入任意寄存器。
   现象为都输出了 `[kernel] IllegalInstruction in application, kernel killed it.`，分析 rcore 中的相应代码，知道了触发了 `Trap::Exception(Exception::IllegalInstruction)` 异常
   其中版本为 RustSBI version 0.3.0-alpha.2, adapting to RISC-V SBI v1.0.0

2.
   1. 在进 `__alltraps` 开始，sp 指向的是用户栈地址，通过 `csrrw sp, sscratch, sp` 使得 sp 指向内核栈地址，__restore 调用的场景：在初始化的时候，调用 __alltraps 后保存上下文；在遇到 trap 的时候从 S mode 切换到 U mode，执行用户态程序

   2. L43 - L48 分别从栈上加载了 t0 - t2 的值，然后分别 store 到 sstatus, sepc, sscratch 里面，其作用分别如下：

      - sstatus 存储特权级控制信息，像是用户态下是否允许中断等
      - sepc 表示用户程序应该返回执行的地址
      - sscratch 保存用户态栈帧，便于恢复上下文

   3. 因为可以对照 `__alltraps` 看，没有保存 t2,t4 是因为 t2 是 sp(用户态栈地址) 被后续 sscratch 保存了，t4(tp) 不被用户态程序使用，所以不需要保存，也不需要恢复了

   4. 该操作交换 sp 和 sscratch 里面的值，把 sp 恢复为用户态栈地址，sscratch 留成内核态栈地址

   5. restore 之后进用户态是在 `sret` 指令之后，因为就是 sret 指令的功能，他把 pc 设成 sepc 的值，也就是我们 trap_handler 里面设的用户态程序的地址，然后按照 SPP 改变当前特权级等

   6. L13 也是交换了 sp 和 sscratch 的值，在这之后，sp 指向了内核栈，sscratch 指向了用户态栈

   7. U 态进 S 态通过 sys_exit 的 syscall 或者发生异常来进

honor code

在完成本次实验的过程（含此前学习的过程）中，我曾分别与 以下各位 就（与本次实验相关的）以下方面做过交流，还在代码中对应的位置以注释形式记录了具体的交流对象及内容：
无

此外，我也参考了 以下资料 ，还在代码中对应的位置以注释形式记录了具体的参考来源及内容：
无

我独立完成了本次实验除以上方面之外的所有工作，包括代码与文档。 我清楚地知道，从以上方面获得的信息在一定程度上降低了实验难度，可能会影响起评分。

我从未使用过他人的代码，不管是原封不动地复制，还是经过了某些等价转换。 我未曾也不会向他人（含此后各届同学）复制或公开我的实验代码，我有义务妥善保管好它们。 我提交至本实验的评测系统的代码，均无意于破坏或妨碍任何计算机系统的正常运转。 我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按“-100”分计。